Rešitev

Najprej preberemo podatke. Tu ni nič takšnega, česar ne bi že videli.

```
import numpy as np

mapping, state = open("example.txt").read().split("\n\n")
mapping = np.array([c == "#" for c in mapping.replace("\n", "")])
state = np.array([[c == "#" for c in v.strip()] for v in state.splitlines()])
```

Zdaj je `mapping` tabela, ki pove, v kakšno stanje se preslika posamezna vsota. Če bomo imeli vsote

```
sums = np.array([[7, 3, 2],
                 [0, 5, 1]])
```

se bo to preslikalo v

```
mapping[sums]
```

```
array([[ True, False,  True],
       [False, False, False]])
```

V zanki bomo počeli tole.

Najprej pripravimo tabelico, ki v sredi vsebuje trenutno stanje, okrog nje pa je dva elementa širok rob, ki vsebuje ničle za sode in enice z lihe korake, saj "neskončna okolica" utripa.

```
i = 0

h, w = state.shape
padded = np.full((h + 4, w + 4), i % 2)
padded[2:-2, 2:-2] = state
```

Matriko smo sestavili z `np.full`, potem pa v sredino prepisali trenutno stanje.

Nato pripravimo tabelico za vsote. Ta bo za dve polji večja od trenutnega stanja, saj se "obljudeni" del tabele vsakič poveča za `1` v vsako smer.

```
sums = np.zeros((h + 2, w + 2), dtype=int)
```

In zdaj je na vrsti seštevanje. Namesto da bi računali vsoto za vsako celico posebej, bomo prištevali `padded` k vsotam, takole:

```
p = 256
for y in range(3):
    for x in range(3):
        sums += p * padded[y:y + h + 2, x:x + w + 2]
        p //= 2
```

`y` in `x` sta odmik od zgornjega in od desnega roba, oba gresta od 0 do 1. `padded[y:y + h + 2, x:x + w + 2]` je torej "okno" v `padded`, ki je enakih

dimenzij kot `sums` in se pomika prek padded. Prištevamo ga k `sums` (kar smemo, saj sta enakih dimenzij. Ob prvem prištevanju (tistem brez odmikov) ga množimo z `256`, ob drugem k `128` in tako naprej.

Kako to deluje - in zakaj deluje - se prepričajte sami. Predstavljajte si nek element `sum` (ne ravno čisto robnega, naj bo nekje na sredi) in razmislite, kateri elementi `padded` se prištejejo vanj ter s kakšnimi faktorji.

Ko dobimo vsote, jih premapiramo čez `mapping`.

```
mapping[sums]
```

```
array([[False,  True,  True, False,  True,  True, False],
       [ True, False, False,  True, False,  True, False],
       [ True,  True, False,  True, False, False,  True],
       [ True,  True,  True,  True, False, False,  True],
       [False,  True, False, False,  True,  True, False],
       [False, False,  True,  True, False, False,  True],
       [False, False, False,  True, False,  True, False]])
```

To je to. Vse skupaj samo še zložimo v zanko.

```
import numpy as np

mapping, state = open("example.txt").read().split("\n\n")
mapping = np.array([c == "#" for c in mapping.replace("\n", "")])
state = np.array([[c == "#" for c in v.strip()] for v in state.splitlines()])

for i in range(50):
    h, w = state.shape
    padded = np.full((h + 4, w + 4), i % 2)
    padded[2:-2, 2:-2] = state
    sums = np.zeros((h + 2, w + 2), dtype=int)
    p = 256
    for y in range(3):
        for x in range(3):
            sums += p * padded[y:h + 2 + y, x:w + 2 + x]
            p //= 2
    state = mapping[sums]
print(np.sum(state))
```

```
5097
```

Kot sem rekel: nič posebnega, samo kup drobnih zafrkancij.

Temu, kar počnemo v tej nalogi, se sicer reče konvolucija. Uporabljamo jo v procesiranju signalov in obdelavi slik, tudi globoke nevronske mreže pogosto vsebujejo konvolucijski nivo, ki počne nekaj podobnega, kot delamo v tej nalogi. Zato ima modul scipy, ki ga pogosto uporabljamo skupaj z numpyjem, funkcijo scipy.signal.convolve2d, ki počne natančno to, kar delamo znotraj te zanke.